An Interactive Environment for Visualizing and Animating Algorithms

Jürgen Döllner, Klaus Hinrichs, Hermann Spiegel

FB 15, Institut für Informatik, Westfälische Wilhelms-Universität
Einsteinstr. 62, D - 48149 Münster, Germany
E-Mail: {dollner | khh | spiegel}@uni-muenster.de

Abstract

This communication describes an interactive environment for visualizing and animating geometric algorithms and geometric data. In contrast to traditional systems for algorithm animation, our approach is based on a new paradigm for modeling geometry and behavior, the *geometry graphs* and *behavior graphs*. They allow developers to specify complex geometric representations of algorithm-specific geometric data, to relate time- and event-dependent processes to geometric data, and to add interactive capabilities to both animation and geometry components. This paradigm reduces the complexity to specify the visualization and animation of an algorithm significantly. Visualization, animation and algorithms are glued together by an interactive environment based on [incr Tcl], an object-oriented extension to Tcl/Tk. The system has been used successfully for interactive animations of computational geometry algorithms, to produce high-quality videos, and as a graphics kernel for 3D applications.

Keywords: Algorithm Animation, Geometric Modeling, Visualization, Computer Graphics, Simulation.

Introduction

Computational geometry benefits from algorithm visualization and animation because these techniques allow us to explore how algorithms work, and to see how and when these algorithms transform geometric data. Traditional algorithm animation systems are faced by several limitations. Most systems require that an algorithm represents its geometric data in terms of built-in geometric entities which leads to data duplication and a loss of application semantics in the visualization subsystem. Another limitation results from a tight coupling to a specific rendering toolkit. In this case, we cannot exchange, for example, a fast real-time graphics engine (e.g., used for interactive simulations) with a high-quality photorealistic renderer (e.g., used for video production).

The description of time- and event-dependent behavior of algorithms and geometric data is of considerable complexity. Most (procedural) approaches to algorithm animation cannot reflect the dynamic relationships between algorithms and geometric data in an object-oriented fashion.

Computational Geometry '97 Nice France

In addition, algorithm animation is still limited with respect to 3D interaction capabilities since 3D interaction requires input techniques which are typically less developed in the underlying low-level 3D graphics systems. However, 3D interaction is the key to interactive algorithm animations which offer new strategies for exploring the dynamic behavior of algorithms (e.g., modification of input data in real-time).

Another more technical issue is the integration of algorithm libraries into the animation system. If an animation system forces the re-coding of an algorithm, or if it does not link to state-of-the-art user interface technology, developers will not use it. There are also competing architecture requirements: graphics and algorithms should be compiled in order to be fast, whereas interpretative languages make animation design, algorithm test, and user interface development easier. Compared to existing systems for algorithm animation (e.g., GASP [12] or Obliq-3D [8]), our system focusses on an explicit time management, 3D interaction techniques, and an extendible 3D graphics kernel.

System Architecture

The architecture of our system consists of two layers, the MAM modeling layer, and the VRS rendering layer (Fig. 1). The *Modeling and Animation Machine* [3] is responsible for the geometric modeling and the modeling of animation and interaction. The *Virtual Rendering System* [2] provides a uniform object-oriented interface to different low-level 3D rendering toolkits, e.g. OpenGL, Radiance, POVRay and RenderMan. MAM/VRS operates on shared geometric and graphics objects which represent algorithm (i.e., application) data. Objects from all subsystems can be accessed by an [incr Tcl] [11] shell.

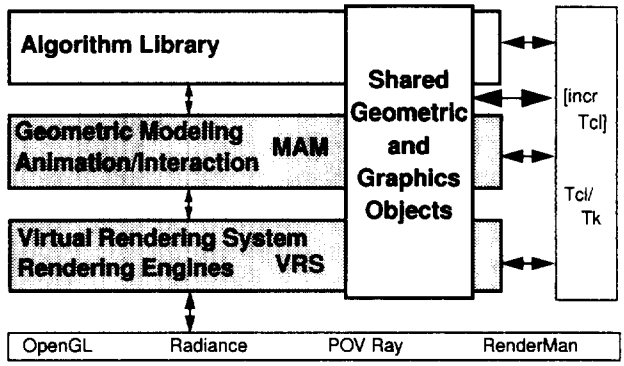

Figure 1: MAM/VRS System Architecture.

Integration of Geometric Data

One of the key differences to existing systems is that we explicitly encourage the usage of algorithm-specific data types in the modeling and rendering subsystems. There is no notion of "built-in" data types (although a large variety of classes is supplied) since MAM/VRS allows developers to integrate their own data types by inheritance. As a consequence, data duplication is avoided since geometric data need not be converted into built-in graphics primitives, and semantics of data types are available both to animation and rendering. If new algorithm-specific types are integrated into MAM/VRS, they can use the animation features and the scripting language. Geometric and graphics objects are associated with *geometry nodes* which are organized in directed acyclic *geometry graphs*.

In our experience, an integrated solid modeler has been a valuable tool for algorithm animation. For example, it allows to explore polyhedral objects using solid operators.

Example: Regular Polyhedron with Hidden Symmetries

The polyhedral realization of Coxeter's {4,6|3} exposes hidden symmetries [1]. To inspect the geometric structure of the polyhedron, we project the edges of the polyhedron onto a canvas (Fig. 2), and we allow the user to split the polyhedron by an arbitrary plane. During the rotation of the polyhedron, its projected edges are updated.

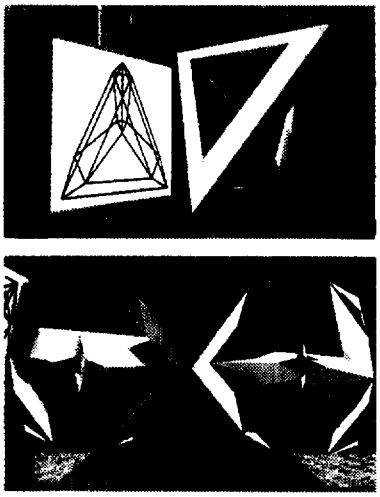

Figure 2: Polyhedron and Edge Projection. Splitted Polyhedron.

Solid modeling operations are essential to achieve constructive geometric results.

Example: 3D Voronoi Cells

We use an interactive application to construct and explore 3D Voronoi cells. The cells are created when new 3D points are inserted into a default solid volume by split operations. The cell complex can be decomposed interactively, and a clipping plane can be used to extract cross sections (Fig. 3).

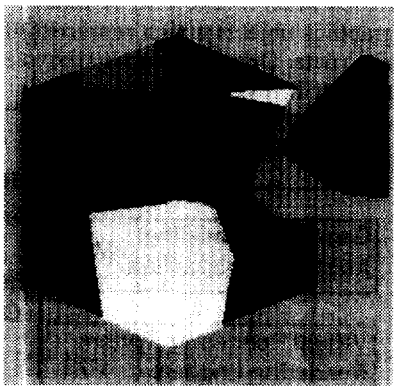

Figure 3: 3D Voronoi Cell Inspector.

Animation of Algorithms

Algorithm animations are specified by directed acyclic graphs called *behavior graphs*. They control the flow of time and events, constrain geometric and graphics objects, and can be used to code story books. Geometry and behavior graphs are tightly coupled since both are associated with shared geometric and graphics objects (Fig. 4). Behavior nodes calculate, distribute and transform model time and time intervals. Each geometric object has its time requirement and gets synchronized when a new point in time (of its local model time) is reached.

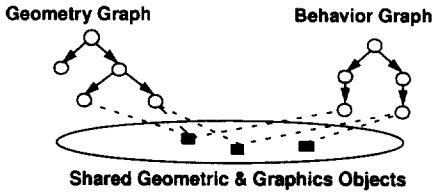

Figure 4: Object sharing between the graphs.

For example, to translate a 3D point onto a paraboloid, we associate the point with a behavior node which constrains its position, and assign a time requirement (i.e, the desired duration) to the behavior node. If we activate the behavior node, a time process is started and sends synchronization events to the behavior node which in turn calculates and updates the position of the point.

Example: Constructing a Voronoi Diagram

For a set of n points in a d-dimensional Euclidean space, the arrangement induced by these sites is defined by the n hyperplanes that are tangential to the $(d+1)$-dimensional unit parabola at the points lifted to the parabola [5]. For $d=2$ we can visually construct the 2-dimensional Voronoi diagram as follows: (1) the points are lifted from the xy-plane onto the unit paraboloid, (2) planes that are tangential to the lifted points on the paraboloid are faded in, (3) the plane and the paraboloid are faded out, and finally the Voronoi diagram is exposed (Fig. 5) by looking from above into the open paraboloid.

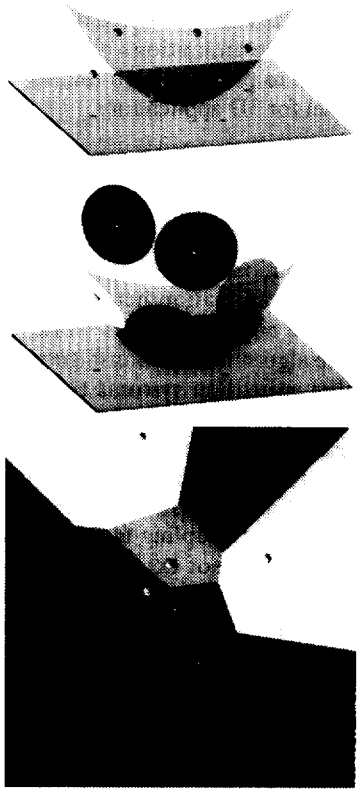

Figure 5: Tangential Planes Constructing the Voronoi Diagram.

For each point we create a sphere object and a tangential plane object. In (1), we constrain the sphere position of each site by a translation constraint behavior node. The speed is specified relative to the distance to the paraboloid, and expressed by the time requirements of

the behavior nodes. The growing of the tangential planes is represented by behavior nodes which scale the facet. Fade-in and fade-out effects are achieved by constraining the transmission attributes of the *xy*-plane and the paraboloid.

We observed that the time management facilitates the design of large and complex animations since we can use a variety of basic behavior types which can be combined and specialized, instead of purely functional descriptions [3]. For example, time manipulator nodes can be mixed arbitrarily with constraint behavior nodes. For example, to slow down the lifting of a sphere close to its final position, we prefix its behavior node with a "creep" behavior node.

Interactive Animations

Interaction nodes control the flow of events in behavior graphs and manipulate associated shared geometric and graphics objects. Interaction nodes are specialized behavior nodes and contained in behavior graphs. They can be combined with time-related behavior nodes and form part of animations. For example, while a sweep plane is moving, the 3D scene can be rotated by a trackball.

Interaction can be used in algorithm animations to configure geometric data, to control and design the animation steps, and to observe the execution of the algorithm. One limitation of many (algorithm) animation systems is that they do not offer *three-dimensional* and *precise* interaction mechanisms. For example, during the motion of a point aligned to a plane, the interaction mechanism must ensure that the mouse position and the associated 3D position coincide, and that the point remains constrained to the plane.

MAM/VRS offers a collection of interaction classes which can be customized to algorithm-specific geometric data, (e.g. 3D translators constrained to lines or planes) and selection mechanisms for arbitrarily contoured selection regions. The orthogonality of geometry nodes and behavior nodes, and their association to a pool of shared application objects provides a clear organization scheme for animations.

Example: Interactive Animation of Fortune's Algorithm

Fortune [6] developed an optimal algorithm for constructing the Voronoi diagram by sweeping the plane with a parabolic front [9]. In our animation we create for each point a sphere and a cone; the cone is erected upon the point. The user can drag interactively the points, and see how the Voronoi diagram changes. The interaction handler must guarantee that each point is not dragged outside the *xy*-plane. The camera can be animated to obtain the "best view" from $z = -\infty$. Additionally, the parabolic front is obtained by intersecting a 45-degree plane with the cones. The user can sweep the plane along the *x*-axis. Fig. 6 (left) shows the 3D configuration; Fig. 6 (right top) shows the parabolic front, and Fig. 6 (right bottom) exposes the Voronoi diagram (the semi-transparent circular disk which embeds the given points is provided as help for orientation).

Implementation

The system is implemented in C++. We provide a C++ and an [incr Tcl] application programming interface. The mapping between C++ and [incr Tcl] is based on [7]. Bindings for different user interface toolkits are available (e.g., Motif, ViewKit, Tk [10]). Algorithm animations can be previewed and designed with the OpenGL rendering engine. To produce a high-quality video, MAM/VRS supports among different renderers Pixar's RenderMan. A native Java implementation is under development.

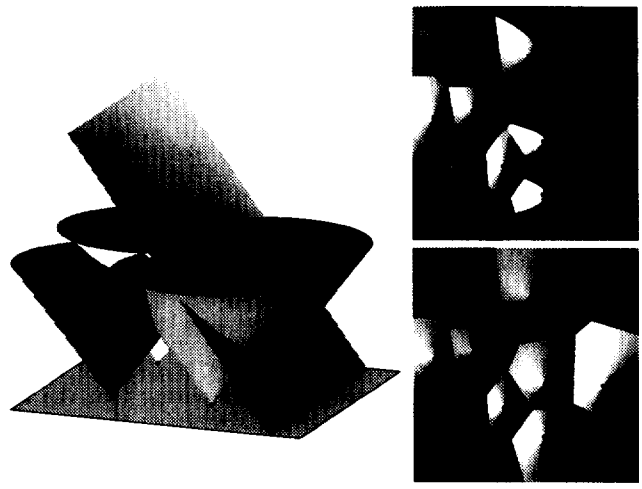

Figure 6: Interactive Animation of Fortune's Algorithm.

Some animations and more details about MAM/VRS can be found at `http://wwwmath.uni-muenster.de/~mam`.

References

1. J. Bokowski, J. M. Wills: *Regular Polyhedra with Hidden Symmetries*, The Mathematical Intelligencer, **10**(1), 27-32 (1988).
2. J. Döllner, K. Hinrichs: *The Virtual Rendering System - A Toolkit for Objet-Oriented 3D Rendering*, EduGraphics/CompuGraphics, Combined Proceedings, 309-318, 1995.
3. J. Döllner, K. Hinrichs: *Object-Oriented 3D Modeling, Animation and Interaction*, The Journal of Visualization and Computer Animation, **8**(1), 33 - 64 (1997).
4. J. Döllner: *Object-Oriented 3D Modeling, Animation and Interaction*, PhD Thesis, University of Münster, 1996.
5. H. Edelsbrunner, R. Seidel: *Voronoi diagrams and arrangements*, Discrete & Computational Geometry, 1, 25-44 (1986).
6. S. Fortune: *A Sweepline Algorithm for Voronoi Diagrams*, Algorithmica **2**, 153-174 (1987).
7. W. Heidrich, P. Slusallek, H.-P. Seidel: *Using C++ Class Libraries from an Interpreted Language*, Technology of Object-Oriented Languages & Systems, TOOLS 14 (Santa Barbara), 397-408, 1994.
8. M. A. Najork, M. H. Brown, *Obliq-3D: A High-Level, Fast-Turnaround 3D Animation System*. IEEE Transactions on Visualization and Computer Graphics, **1**(2), 175-192 (1995).
9. J. O'Rourke. Computational Geometry in C, Cambridge University Press, 1994.
10. J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.
11. M. J. McLennan, *[incr Tcl]: Object-Oriented Programming in Tcl*, Proceedings of the Tcl/Tk Workshop 1993, University of California, http://www.wn.com/biz/itcl.
12. A. Tal and D. Dobkin *Visualization of Geometric Algorithms*, IEEE Transactions on Visualization and Computer Graphics, **1**(2), 194-204 (1995).